FIG. 1
PRIOR ART

```
1    c:\collections
2        notes.txt
3        myletter.doc
4        c-myhomepage
5
6          s
7            homepage.html
8            myphoto.jpg
```

FIG. 2

```
1    c:\collections
2        notes.txt
3        myletter.doc
```

100

```
4        c-myhomepage
5          cspec
6          s
7            homepage.html
8            myphoto.jpg
```

FIG. 3

102

```
1    collection        c-myhomepage
2    coll-type         cf-web-page
3    coll-desc         A sample homepage collection
4    end-collection
```

FIG. 4

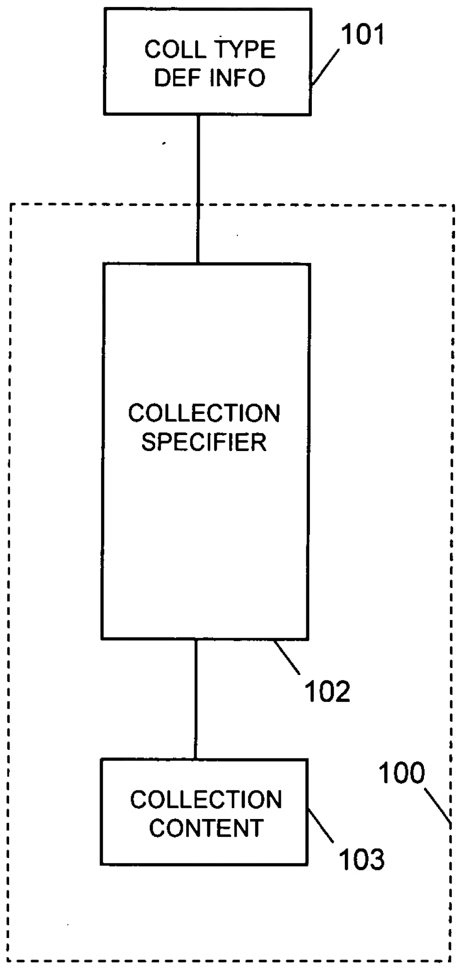

COLL TYPE DEF INFO
101
COLLECTION SPECIFIER
102
COLLECTION CONTENT
103
100

FIG. 5

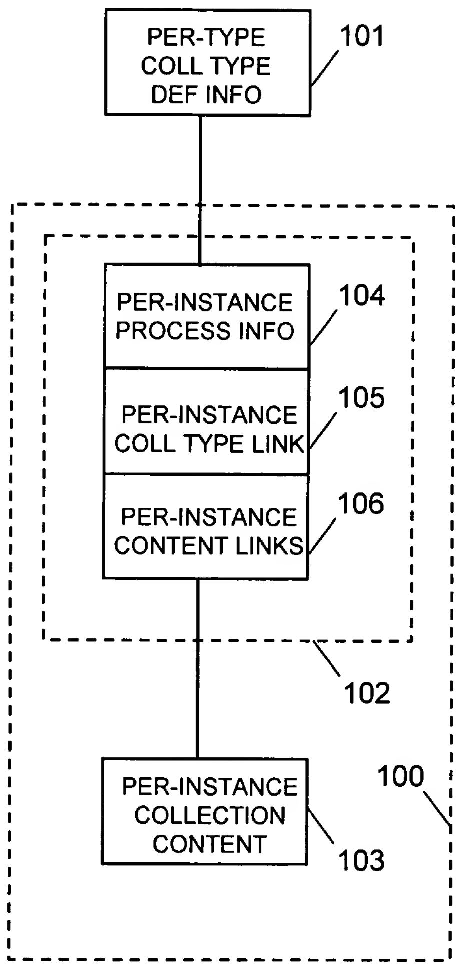
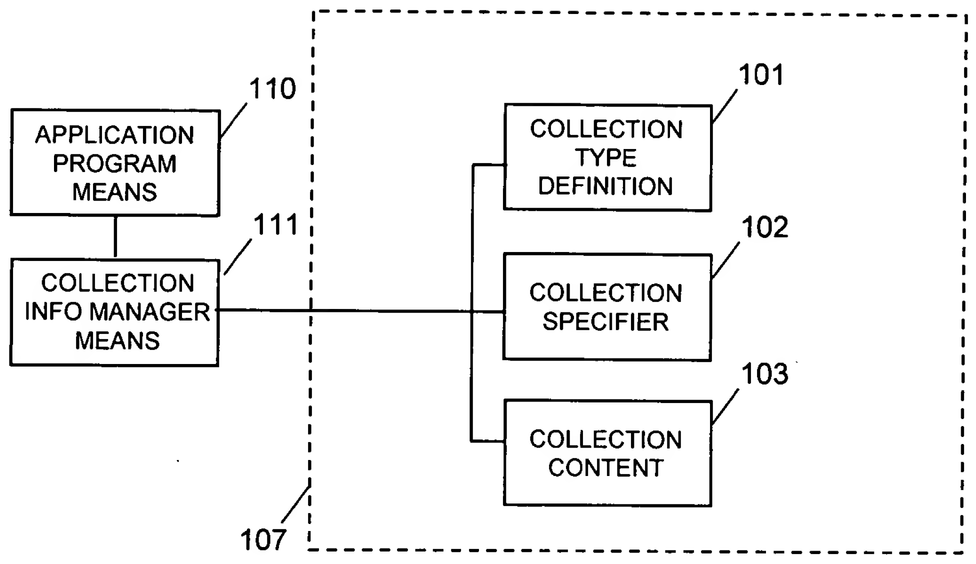
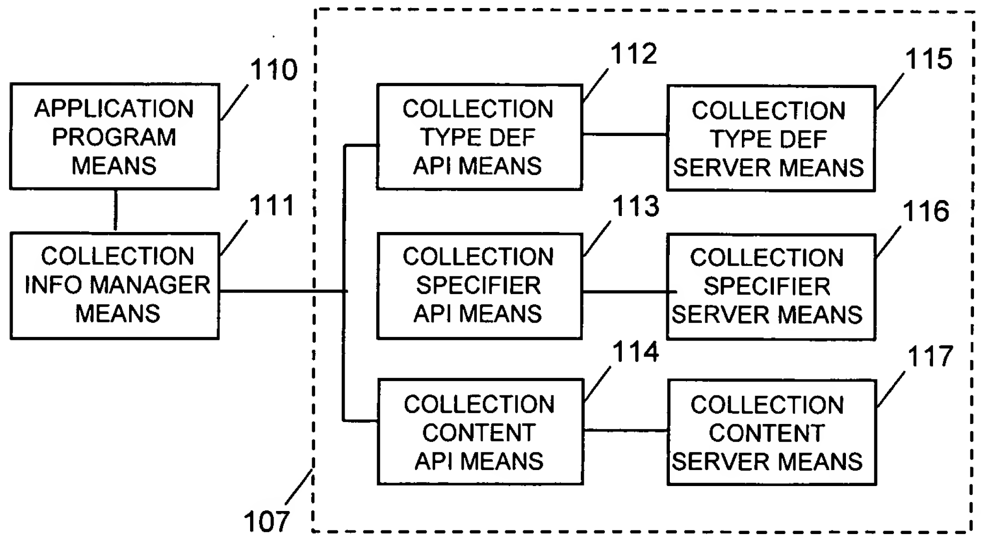

PER-TYPE COLL TYPE DEF INFO
101
PER-INSTANCE PROCESS INFO
104
PER-INSTANCE COLL TYPE LINK
105
PER-INSTANCE CONTENT LINKS
106
102
PER-INSTANCE COLLECTION CONTENT
103
100

FIG. 6



110
APPLICATION PROGRAM MEANS
111
COLLECTION INFO MANAGER MEANS
101
COLLECTION TYPE DEFINITION
102
COLLECTION SPECIFIER
103
COLLECTION CONTENT
107

FIG. 7



110
APPLICATION PROGRAM MEANS
111
COLLECTION INFO MANAGER MEANS
112
COLLECTION TYPE DEF API MEANS
115
COLLECTION TYPE DEF SERVER MEANS
113
COLLECTION SPECIFIER API MEANS
116
COLLECTION SPECIFIER SERVER MEANS
114
COLLECTION CONTENT API MEANS
117
COLLECTION CONTENT SERVER MEANS
107

FIG. 8

```
/* collection data structure */
collection-info {

  + specifier_info
      + coll-type-indicator
      + other specifier information ...

  + content_info
      + content_location_info ...
      + content_members ...
      + other content information...

  + other collection structure information...
}
```

FIG. 9

```
/* collection type definition data structure */
collection-type-definition-info {

  + coll-type-name
  + collection internal structure info ...
  + collection content location info ...
  + collection content type recognition info ...

  + other collection type definition information...
}
```

FIG. 10

	KEY	VALUE	
1	/* collection type internal structure definitions */		
2	dir_source_files	./s	
3	dir_doc_files	./doc	
4	/* content location definitions (per-type content links) */		
5	content_subtree_http	http://host.com/some/dir/name	
6	content_subtree_ftp	ftp://host.com/some/dir/name	
7	content_subtree_nfs	/some/local/directory/name	
8	/* content type recognition definitions */		
9	content_policy	subtree_below_cspec_file	
10	content_file_type	.c	file_cpp
11	content_file_type	.c	file_c
12	content_file_type	.h	file_c_include
13	content_file_type	.doc	file_ms_word
14	content_file_type	.html	file_html
15	content_file_type	.xls	file_ms_excel
16	/* collection processing definitions */		
17	compile_c_files	yes	
18	compiler_windows	vc++	
19	compiler_unix	gcc	
20	build platforms	Win98, Win2000, gnulinux	
21	process files	compile link	
22	link libraries	stdio math sock	
23	/* results dispatching definitions */		
24	results_ftp_host	ftp.output.com	
25	results_ftp_dir	c:\ftphome\collection\results	

FIG. 11

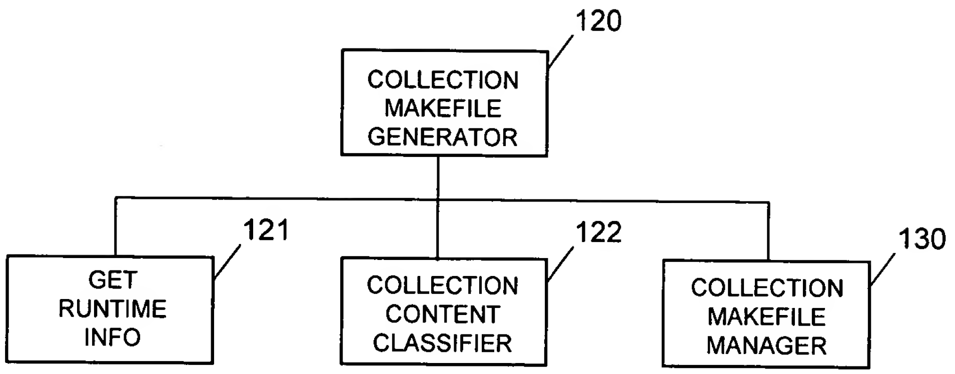

COLLECTION MAKEFILE GENERATOR
120
GET RUNTIME INFO
121
COLLECTION CONTENT CLASSIFIER
122
COLLECTION MAKEFILE MANAGER
130

FIG. 12

```
1  /* simplified algorithm for collection makefile generator */
2  Call get runtime info to get invocation parameters
3  Call collection content classifier to classify collection content
4  Call collection makefile generator manager to generate
   a complete makefile, passing classifier information as input
```

FIG. 13

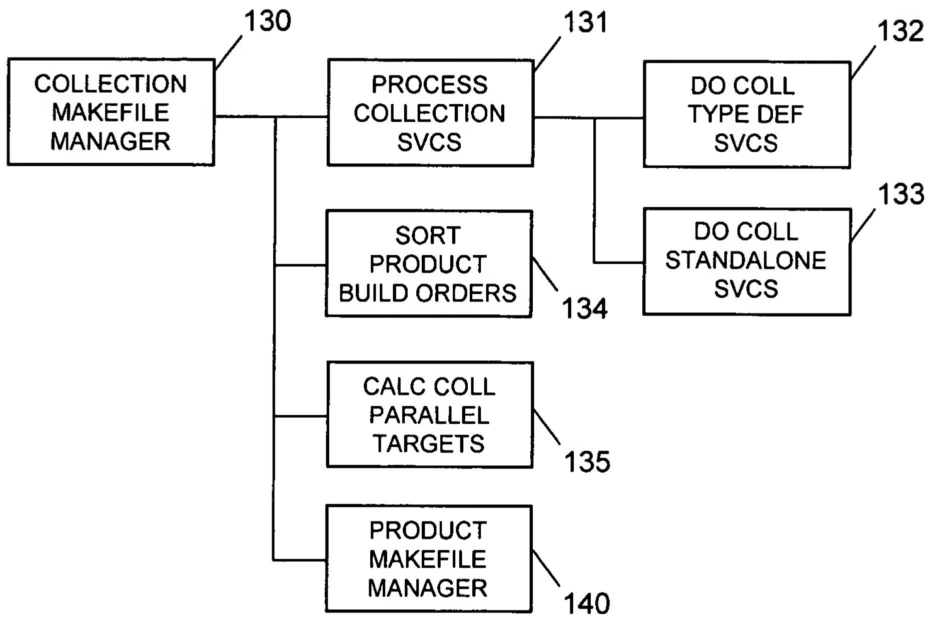


FIG. 14

```
/* simplified algorithm for collection makefile manager */
Process collection-level fragments
    Process fragments from collection type definition
    Process fragments from collection specifier
Determine relative build order among multiple products
Determine number, names of coll-level parallel build targets
Loop over each product in collection
    Process each product by calling product makefile manager
```

FIG. 15

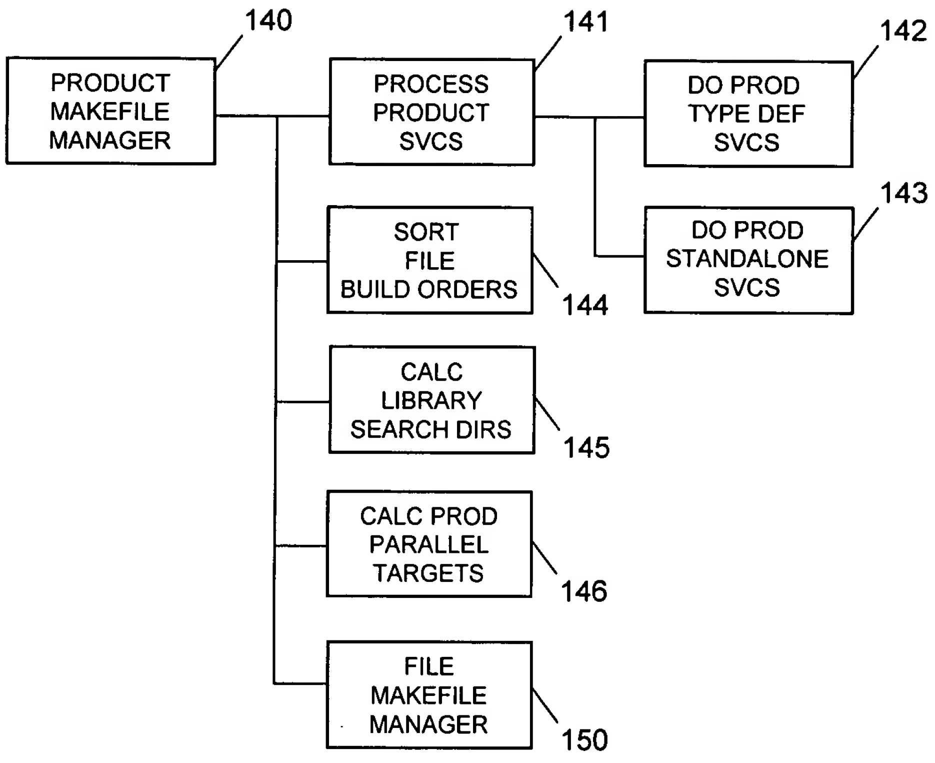


FIG. 16

```
1   /* simplified algorithm for processing one product */
2   Process product-level fragments
3       Process fragments from product  type definition
4       Process fragments from product section of collection specifier
5   Determine relative build order among content files for product
6   Determine number, names of product-level parallel build targets
7   Loop over each content file
8       Process each content file by calling file makefile manager
```

FIG. 17

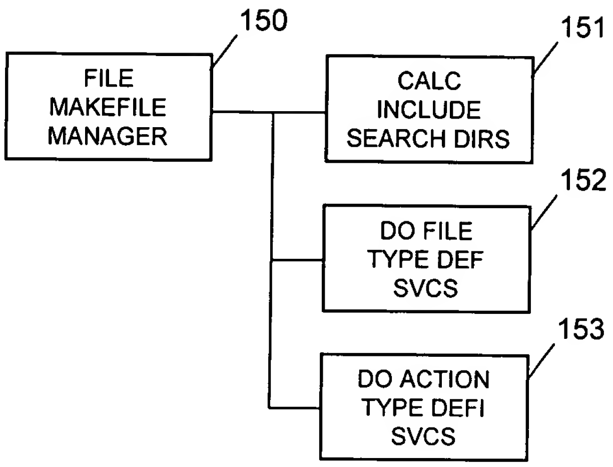


FIG. 18

```
1   /* simplified algorithm for processing one content file */
2   Calculate include file search directories
3   Process fragments from content  type definition
4   Process fragments from action type definition
```

FIG. 19

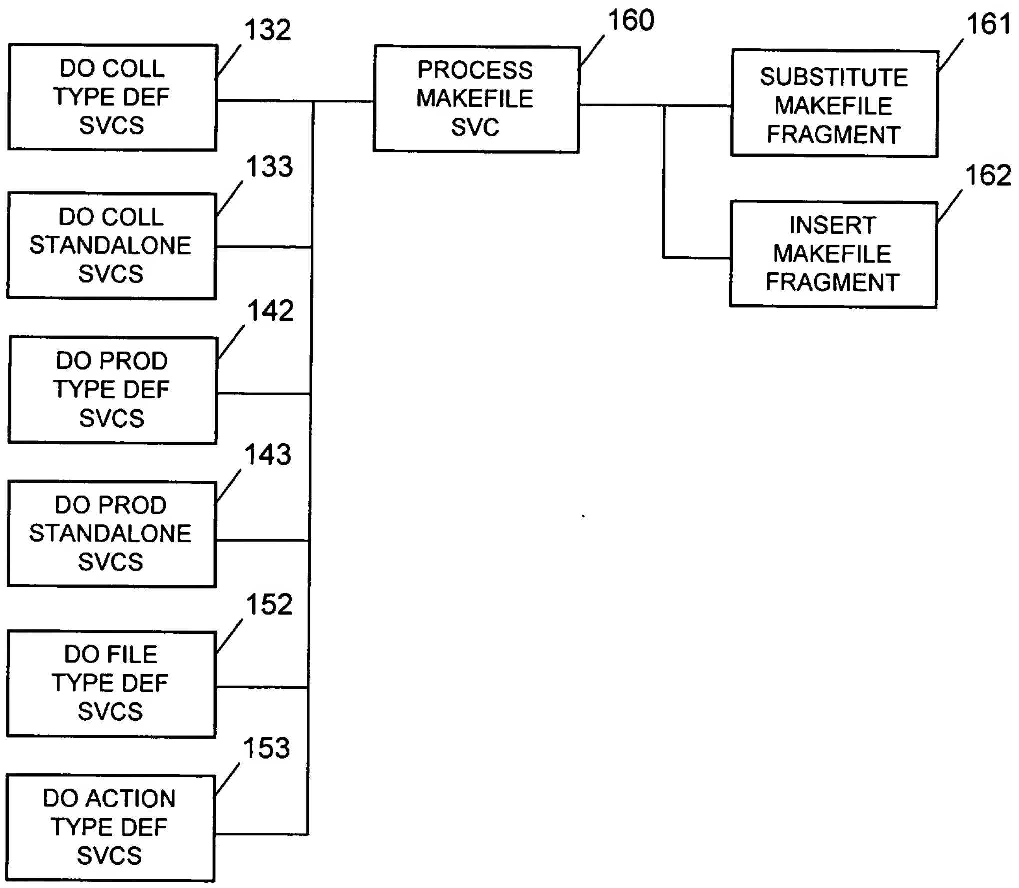


FIG. 20

```
1    /* simplified algorithm for processing one fragment */
2    Substitute replacement values for placeholder strings
3    Insert substituted fragment into makefile data structure
```

FIG. 21

```
c:\collections
    c-my-example
        cspec
        s
            pi
                cmdline.h
            win98
                cmdline.c
            gnulinux2
                cmdline.c
        lib
            pi
                libfuns.h
                libfuns.c
```

FIG. 22

```
cspec:

collection   c-my-example
coll-type    ct-program
coll-desc    A multi-platform C program with library.
end-collection

product      myprog
prod-type    pt-program
prod-desc    A program product.
libs         team-lib gnulinux-lib
end-product

product      mylibrary
prod-type    pt-library
prod-desc    A library product.
end-product
```

FIG. 23

```
/* classification output for gnulinux2 platform */
collection          c-my-example
coll-type           ct-program
... other coll classification info


/* classification info for a program product */
product             myprog
prod-type           pt-program

content             cmdline.h
content-path        ../s/pi/cmdline.h
content-type        ctype-c-header
content-language    c
end-content

content             cmdline.c
content-path        ../s/gnulinux2/cmdline.c
content-type        ctype-c-source
content-language    c
content-dep         ../s/pi/cmdline.h
content-dep         ../lib/pi/libfuns.h
content-dep         external-incl-file.h
content-dep         team-incl.h
end-content

end-product
```

FIG. 24

```
1   /* classification output  */
2   collection              c-my-example

3   ... /* classification info for the host collection */
4   ... /* classification info for the program product */

5   /* classification info for a library product */
6   product                 mylibrary
7   prod-type               pt-library

10  content                 libfuns.h
11  content-path            ../lib/pi/libfuns.h
12  content-type            ctype-c-header
13  content-language        c
15  end-content

16  content                 libfuns.c
17  content-path            ../lib/pi/cmdline.c
18  content-type            ctype-c-source
19  content-language        c
21  content-dep             ../lib/pi/libfuns.h
22  end-content

23  end-product
```

FIG. 25

```
 1  collection type definition information
 2    product type definition information
 3      content type definition information
 4         action type definition information


 5  cspec:
 6  coll-type                  ct-program

 7  index-coll-types.tbl:
 8  ct-program                 ct-program.def
 9  ct-web-page                ct-web-page.def

10  ct-program.def:
11  product-type-index         index-product-types.tbl

12  index-product-types.tbl:
13  pt-program                 pt-program.def

14  pt-program.def:
15  content-type-index         index-content-types.tbl

16  index-content-types.tbl
17  ctype-c-source             content-c.def

18  content-c.def:
19  action-type-index          index-action-types.tbl

20  index-action-types.def:
21  action-c-source            action-c-source.def

22  action-c-source.def:
23  ... action definition information
```

FIG. 26

```
index-coll-types.tbl:
ct-program                ct-program.def
ct-library                ct-library.def
ct-doc-html               ct-html.def


ct-program.def:
/* type definition info for a "ct-program" collection type */
product-type-index        index-prod-program.tbl

base-template             base-template.tpl

service                   svc-coll-macro-platform
service                   svc-coll-macro-site
service                   svc-coll-macro-tool-names
service                   svc-coll-macro-compiler
service                   svc-coll-macro-suffix
service                   svc-coll-target-defaults
service                   svc-coll-target-others

parallelism-max           4

... other collection type info
```

FIG. 27

```
 1  index-prod-program.tbl:
 2  pt-program                        pt-program.def
 3  pt-program-java                   pt-program-java.def
 4  pt-program-unix                   pt-program-unix.def
 5  pt-program-win                    pt-program-win.def


 6  pt-program.def:
 7  /* type definition info for a "program" product type */
 8  dir-source-files                  dirs-source.lst
 9  dir-library-files                 dirs-library.lst
10  dir-include-files                 dirs-include.lst
11  file-identification-table         file-identification.tbl
12  content-type-index                index-content-types.tbl
13  service                           svc-prod-program
14  ... other product type info
```

FIG. 28

```
index-content-types.tbl:
ctype-c-source              content-c.def
ctype-c-header              content-c-h.def
ctype-csh                   content-csh.def
ctype-html                  content-html.def

content-c.def:
/* type definition info for a “c” file type */
type                        c-source
language                    c
action                      action-c-source
action-type-index           index-action-types.tbl
service                     svc-file-c-source
... other content type definition info
```

FIG. 29

```
index-action-types.tbl:
action-c-source             action-c-source.def
action-c-header             action-c-header.def
action-csh                  action-csh.def
action-html                 action-html.def

action-c-source.def:
parser-type                 internal
parser-name                 internal-c
service                     svc-action-c-source
```

FIG. 30

```
idx-makefile-services.tbl:

/* services for collections */
svc-coll-macro-platform      coll-macro-platform.tpl
svc-coll-macro-site          coll-macro-site.tpl
svc-coll-macro-compiler      coll-macro-compiler.tpl
svc-coll-macro-toolnames     coll-macro-toolnames.tpl
svc-coll-macro-file-suffix   coll-macro-file-suffix.tpl
svc-coll-target-defaults     coll-target-defaults.tpl
...
/* services for products */
svc-prod-program             prod_prog_pi.tpl
svc-prod-program             prod_prog_os.tpl
svc-prod-program             prod_prog_pd.tpl
...
svc-prod-library             prod-lib-pi.tpl
svc-prod-library             prod-lib-os.tpl
svc-prod-library             prod-lib-pd.tpl
...
/* services for files */
svc-file-c-source            file-c.tpl
svc-file-c-header            file-c-header.tpl
svc-file-f90                 file-f90.tpl
svc-file-f90-header          file-f90-header.tpl
svc-file-f90-module          file-f90-module.tpl
...
/* services for actions */
svc-action-c-source          action-c-source.tpl
...
/* services for application tasks */
svc-app-chmod                app-chmod.tpl
svc-app-copy-file            app-copy-file.tpl
...
```

FIG. 31

```
coll-macro-platform.tpl:
# This file defines platform-specific makefile macros

fragment-begin
_marker_ marker-htree copy
# The holding area for shared files and libraries
HTREE=/site/h
fragment-end

fragment-begin
_marker_ marker-macros1 copy
# makefile platform name, virtual platform name
MP=win98.plt
VP=win98
fragment-end
```

FIG. 32

```
coll-macro-site.tpl:
# This file defines site-specific makefile macros
fragment-begin
_marker_ marker-macros1 copy

# places where shared files go
SHARE_DIR=$(HTREE)\share

# places where web pages go
HOST_WEB=www.your_domain.com
...
fragment-end
```

FIG. 33

```
coll-macro-toolnames.tpl:
# define macros for various program names
fragment-begin
_marker_ marker-macros1 copy

LS=ls
DIR=dir
RM=rm
CP=cp
ZIP=zip
UNZIP=unzip
CC=gcc
LIB=ld
RMDIR=rm
fragment-end
```

FIG. 34

```
coll-macro-compiler.tpl:
# This file defines compiler options
fragment-begin
_marker_ marker-macros1 copy

# default compiler options
OPT=
DEBUG=
# default linker options
LIBSPATH = $(HTREE)/$(MP)
LDFLAGS= -s
LPP= -L

fragment-end
```

FIG. 35

```
coll-macro-suffix.tpl:
# defines macros for file suffixes for this platform
fragment-begin
_marker_ marker-macros1 copy

# objects, executables, libraries, archives
O=.o
SO=.so
X=
L=.a
A=
AWKS=.awk
SEDS=.sed
LEXS=.l
YACS=.y
CLASS=.class
fragment-end
```

FIG. 36

```
coll-target-defaults.tpl:
# This file defines default makefile targets
fragment-begin
_marker_ marker-targets0 copy

# default targets used by all makefiles
default: build

all: build exports

build:

exports:
fragment-end
```

FIG. 37

```
/* fragment commands */
fragment-begin / fragment-end
_marker_   marker-name   copy
_macro_    macro-name    append      value1 value2...
_target_   target-name   add-deps    dep1 dep2 ...
_target_   target-name   copy
_target_   target-name   copy-force
```

FIG. 38

```
base-template.tpl:

# marker-htree

# marker-macros1

# marker-targets0
```

FIG. 39

```
makefile.out:

# The holding area for shared files and libraries
HTREE=/site/h
# marker-htree

# makefile platform name, virtual platform name
MP=win98.plt
VP=win98

# places where shared files go
SHARE_DIR=$(HTREE)\share
...
LS=ls
DIR=dir
...
OPT=
DEBUG=
...
O=.o
SO=.so
X=
...
# marker-macros1

# default targets used by all makefiles
default: build

all: build exports

build:

exports:
# marker-targets0
```

FIG. 40

```
prod-prog-pi.tpl:
# Define platform-independent macros for programs

fragment-begin
_marker_ marker-macros1 copy
# Initialize these macros so they are defined.
ALL_OBJS__prod_=
OBJ_PI__prod_=
OBJ_F90__prod_=
OBJ_F90_MOD__prod_=

# create one macro to hold all objects
ALL_OBJS__prod_=$(OBJ_PI__prod_) \
    $(OBJ_F90__prod_) $(OBJ_F90_MOD_prod_)

# add marker to anchor linker  macro later
# marker-link-cmd
fragment-end
```

FIG. 41

```
prod-prog-os.tpl:
# Define operating system macros for programs

# Adds program name dependency to build target.
fragment-begin
_target_ build          add_deps _mprod_$(X)
fragment-end

# Adds program name dependency to export target
fragment-begin
_target_ exports        add_deps _mprod_$(X)
fragment-end
```

FIG. 42

```
prod-prog-pd.tpl:
# Define platform-dependent macros for programs

fragment-begin
_marker_ marker-macros1 copy
# default compiler flags for this platform
CCFLAGS1= -Wall -ansi -pipe -I.
CCFLAGS2= -I- -c
fragment-end

fragment-begin
_marker_ marker-link-cmd copy
# linker command for this platform
LDLIBS=
LD__prod_=${CC} -o _mprod_  _lib_dirs_ \
            $(ALL_OBJS__prod_) _lib_names_
fragment-end

fragment-begin
# add link command to target for program product
_target_ _mprod_$(X)  copy
    $(LD__prod_) $(LDFLAGS)
    $(CHMOD) 775 _mprod_$(X)
fragment-end

fragment-begin
# add object dependencies to product target
_target_ _mprod_$(X) add_deps $(OBJ_PI__prod_)
fragment-end
```

FIG. 43

1	_prod_	name of product from cspec
2	_mprod_	name of product file on disk
3	_ptype_	product type of current product
4	_src_file_path_	source file pathname
5	_src_file_name_	source file filename
6	_src_file_name_no_suf_	source filename with no suffix
7	_target_list_	list of makefile targets
8	_target_name_	name of current target
9	_deplist_	list of dependent targets
10	_incl_dirs_	list of include directories
11	_lib_dirs_	list of library directories
12	_lib_names_	list of library names
13	_zpln_	parallel target number 01,02,etc

FIG. 44

```
makefile.out:
...
# Initialize these macros so they are defined.
ALL_OBJS_myprog=
OBJ_PI_myprog=
OBJ_F90_myprog=
OBJ_F90_MOD_myprog=

# create one macro to hold all objects
ALL_OBJS_myprog=$(OBJ_PI_myprog) \
    $(OBJ_F90_myprog) $(OBJ_F90_MOD_myprog)

# marker-link-cmd

# marker-macros1

# default targets used by all makefiles
default: build

all: build exports

build: myprog

exports:  myprog
# marker-targets0
```

FIG. 45

```
makefile.out:
...
# Initialize these macros so they are defined.
ALL_OBJS_myprog=
OBJ_PI_myprog=
...
# create one macro to hold all objects
ALL_OBJS_myprog=$(OBJ_PI_myprog) ...
...
# linker command for this platform
LDLIBS=
LD_myprog=${CC} -o myprog  $(LDLIBS) \
            $(ALL_OBJS_myprog) $(lb)
# marker-link-cmd
...
# default compiler flags for this platform
CCFLAGS1= -Wall -ansi -pipe -I.
CCFLAGS2= -I- -c
# marker-macros1
...
build: myprog

exports:  myprog
...
# add link command to target for program product
myprog:  $(OBJ_PI_myprog)
    $(LD_myprog) $(LDFLAGS)
    $(CHMOD) 775 myprog
# marker-targets0
```

FIG. 46

```
file-c-source.tpl:
# process files

# add current source file to top src file macro
fragment-begin
_macro_ SRC_C              append _src_file_path_
fragment-end

# add current source file to product source file macro
fragment-begin
_macro_ SRC_C__prod_   append _src_file_path_
fragment-end
```

FIG. 47

```
action-c-source.tpl:
# process files

# add compilation command under C object targets.
fragment-begin
_target_ _target_name_$(O) copy
    $(CC) $(OPT) $(DEBUG) $(CCFLAGS1) \
        _incl_dirs_ $(CCFLAGS2) _src_file_path_
fragment-end

# add dependency list to C object target.
fragment-begin
_target_ _target_name_$(O)   add_deps _deplist_
fragment-end
```

FIG. 48

```
makefile.out:
...
SRC_C= ../s/gnulinux2/cmdline.c ...
...
SRC_C_prod_= ../s/gnulinux2/cmdline.c ...
...
# default compiler flags for this platform
CCFLAGS1= -Wall -ansi -pipe -I.
CCFLAGS2= -I- -c
# marker-macros1
...
# default targets used by all makefiles
default: build

all: build exports

build: myprog

exports:  myprog
...
cmdline.o:  ../s/pi/cmdline.h ../lib/pi/libfuns.h
    $(CC) $(OPT) $(DEBUG) $(CCFLAGS1) \
        _incl_dirs_ $(CCFLAGS2) ../s/gnulinux2/cmdline.c

...
# marker-targets0
```

FIG. 49

```
 1   collection        c-my-example
 2   coll-type         ct-program
 3   coll-desc         A fileset example
 4   svc               svc-coll-cleanup
 5   end-collection

 6   product           myprog
 7   prod-type         pt-program
 8   libs              mylib
 9   svc               svc-app-copy-file  myprog myprog.bak
10   end-product
```

FIG. 50

```
cspec:
...

product       myprog
prod-type     pt-program
prod-desc     A normal program binary executable.
end-product

product       myprog-2
prod-type     pt-shared-object
prod-desc     A shared object program executable
replace-name  myprog
end-product


_prod_     becomes cspec     name    myprog-so
_mprod_    becomes diskfile  name    myprog

# add link command to target for program product
_mprod_$(X):
    $(LD__prod_) $(LDFLAGS_prod_)
    $(CHMOD) 775 _mprod_$(X)


# link target for product myprog
myprog$(X):
    $(LD_myprog) $(LDFLAGS_myprog)
    $(CHMOD) 775 myprog$(X)


# link target for product myprog-so
myprog$(SO):
    $(LD_myprog-2) $(LDFLAGS_myprog-2)
    $(CHMOD) 775 myprog$(SO)
```

FIG. 51

```
product-build-order.tbl:
# define relative build order among products

pt-initial          10
pt-data             50
pt-library          100
pt-program          1000
pt-script           1000
```

FIG. 52

```
makefile.out:
...
# dependent targets mylib and myprog appear in proper
# product build order, from left to right
#
build: mylib myprog

mylib:
...
myprog:
...
```

FIG. 53

```
file-build-order.tbl:
# define relative build order among file types

ft-resource           10
ft-precompiled-cpp    20
ft-c-source           50
```

FIG. 54

```
makefile.out:
...
# dependent targets mylib and myprog appear in proper
# product build order, from left to right
#
build: mylib myprog

mylib:
...
myprog: myresource.rc myprecompiled-header.o cmdline.o
...
```

FIG. 55

```
dirs-include.lst:

dir/gnulinux2              /site/myteam/include/gnulinux2
dir/gnulinux2              /site/myteam/include/gnulinux
dir/gnulinux2              /site/include/gnulinux2
dir/gnulinux2              /site/include/gnulinux
```

FIG. 56

```
# suppose these are paths to example include files
/site/include/gnulinux2/external-incl-file.h
/site/myteam/include/gnulinux/team-incl.h

# include files matched by search rules, in order
/site/myteam/include/gnulinux/team-incl.h
/site/include/gnulinux2/external-incl-file.h

_incl_dirs_ = -I /site/myteam/include/gnulinux \
              ... -I /site/include/inux2

makefile.out:
...
file1.o:  ../s/file1.c
    $(CC) $(OPT) $(DEBUG) $(CCFLAGS1) \
    -I /site/myteam/include/gnulinux -I /site/include/inux2 \
    $(CCFLAGS2) ../s/file1.c
```

FIG. 57

```
dirs-library.lst:

dir/gnulinux2          /site/myteam/lib/gnulinux2
dir/gnulinux2          /site/myteam/lib/gnulinux
dir/gnulinux2          /site/lib/gnulinux2
dir/gnulinux2          /site/lib/gnulinux
```

FIG. 58

```
# suppose these are paths to example libraries
/site/lib/gnulinux2/gnulinux-lib.a
/site/myteam/lib/gnulinux/team-lib.a

# libs matched by search rules, in order
/site/myteam/lib/gnulinux/team-lib.a
/site/lib/gnulinux2/gnulinux-lib.a

_lib_dirs = -L /site/myteam/lib/gnulinux –L /site/lib/inux2
_lib_names_ = -l team-lib.a  gnulinux-lib.a

makefile.out:
...
LD_mprog = $(LD)  -L /site/myteam/lib/gnulinux \
             ...  -L /site/lib/gnulinux2 \
             ... -l team-lib.a –l gnulinux-lib.a
...
myprog$(X): ...
   $(LD_mprog) ...
```

FIG. 59

```
virtual-platform.tbl:
#
#                Specific     Generic     Family   Every
# Name           OS           OS          OS       OS
#
gnulinux2.plt    gnulinux2    gnulinux    unix     pi
sol28.plt        sol28        sol         unix     pi
win98.plt        win98        win9        win      pi
win95.plt        win95        win9        win      pi
winnt40.plt      winnt40      winnt       win      pi
win2000.plt      win2000      winnt       win      pi
```

FIG. 60

```
# fragment search directories for win98 platform
fragments/win98
fragments/win9
fragments/win
fragments/pi

# fragment search directories for gnulinux 2 platform
fragments/gnulinux2
fragments/gnulinux
fragments/unix
fragments/pi
```

FIG. 61

```
collection        c-my-example
coll-type         ct-program
coll-desc         A fileset example
end-collection

product           myprog
prod-type         pt-program

libs/pi           mylib
libs/gnulinux     mylib myother-gnulinux-lib

svc/pi            svc-prod-name    svc arguments
svc/gnulinux      svc-prod-name    svc args
svc/win98         svc-prod-name    svc args
end-product
```

FIG. 62

```
makefile.out
...
myprog: file-001.o file-002.o  ... file-100.o
    $(LD_mprog) ...

# GNU make parallelism with –jobs argument will compile
# 4 files at a time to build the myprog target
#
make –j 4 myprog

# without a parallel make tool, makefile targets must be
# generated to offer parallelism, as follows:
#
myprog:  myprog-01 myprog-02 myprog-03 myprog-04
myprog-01: file-001.o file-002.o ...   file-025.o
myprog-02: file-026.o file-027.o ...   file-050.o
myprog-03: file-051.o file-052.o ...   file-075.o
myprog-04: file-076.o file-077.o ...   file-100.o

# now parallel commands can be issued against parallel targets
# running on multiple machines
on machine1:   make myprog-01
on machine2:   make myprog-02
...
# running multiple windows on one machine
in shell window 1: make myprog-01
in shell window 2: make myprog-02
...
# or running in the background on one machine
in shell window 1: make myprog-01 &
in shell window 1: make myprog-02 &
...
```

FIG. 63

```
action-c-source.tpl:
# process files
...
# this line adds the parallelism-specific object file macro to the
# "master" or "top level" object file macro.
fragment-begin
_macro_ OBJ_PI__prod_ append $(OBJ_PI__prod___zpln_)
fragment-end

# this line adds current object file to correct
# parallelism-specific object file macro
fragment-begin
_macro_ OBJ_PI__prod___zpln_ append _target_name_$(O)
fragment-end

# this line adds the parallelism-specific object file macro as a
# dependency of the parallelism-specific build target.
fragment-begin
_target_ build__zpln_ add_deps $(OBJ_PI__prod___zpln_)
fragment-end
```

FIG. 64

```
makefile.out:
...
OBJ_PI_myprog      = file-001.o file-002.o ... file-100.o
OBJ_PI_myprog_01 = file-001.o file-002.o  ... file-025.o
OBJ_PI_myprog_02 = file-026.o file-027.o  ... file-050.o
...
build_01:  $(OBJ_PI_myprog_01)
...
build_02:  $(OBJ_PI_myprog_02)
...
```

FIG. 65

```
makefile.out:
# sequential and parallel targets for multiple products
...
# target for building all products sequentially
build: build_01 build_02 build_03
...
# parallel targets for building all products in parallel
build_01:  myprog-01 product2-01 product3-01 ...
build_02:  myprog-02 product2-02 product3-02 ...
...
# target for building product 'myprog' sequentially
myprog:  myprog-01 myprog-02 myprog-03
...
# parallel targets for building product 'myprog' in parallel
myprog-01: $(OBJ_PI_myprog_01)
myprog-02: $(OBJ_PI_myprog_02)
...
# target for building product 'product2' sequentially
product2: product2-01 product2-02 ...
...
# parallel targets for building product 'product2' in parallel
product2-01:   $(OBJ_PI_product2_01)
product2-02:   $(OBJ_PI_product2_02)
...
```